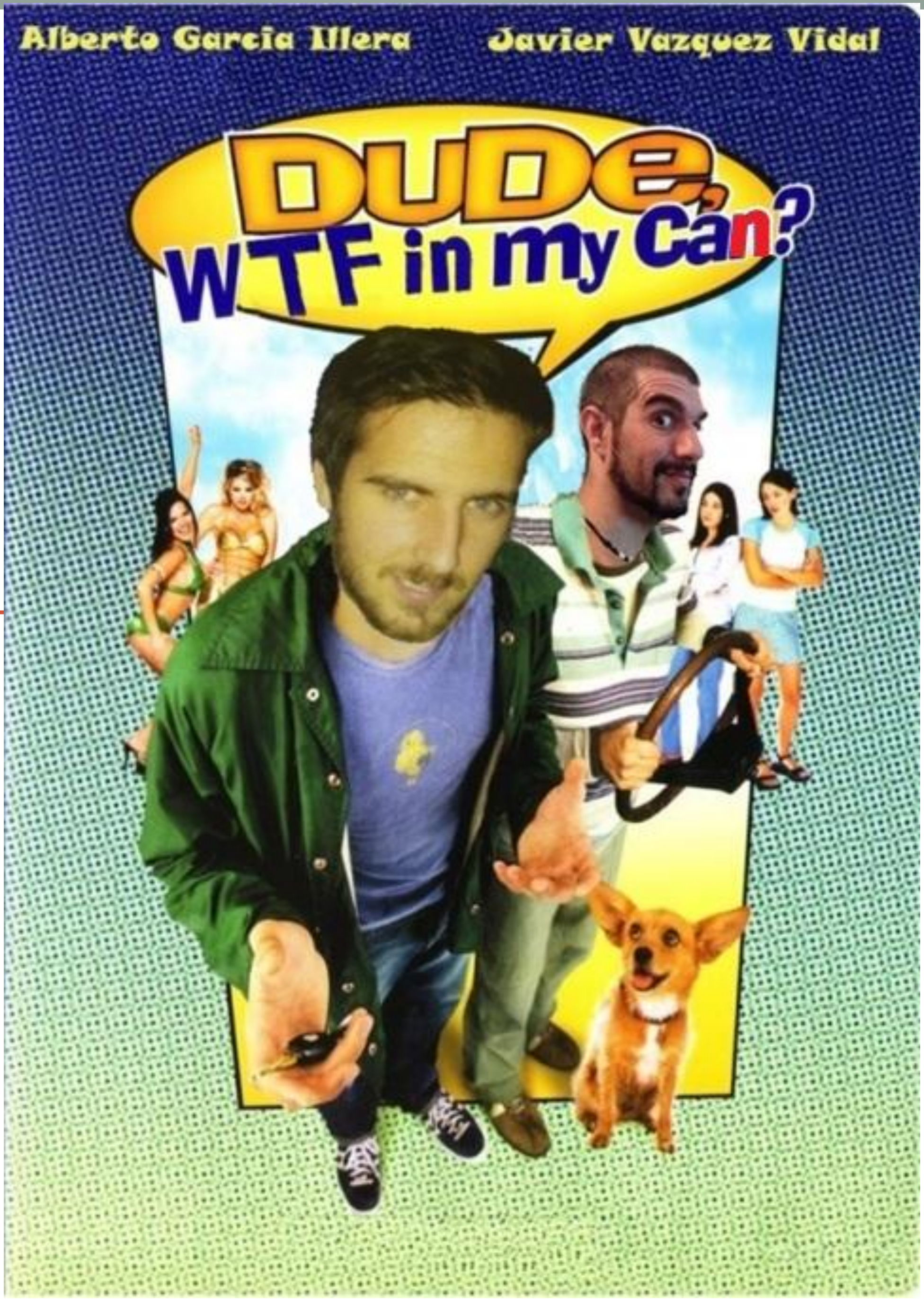
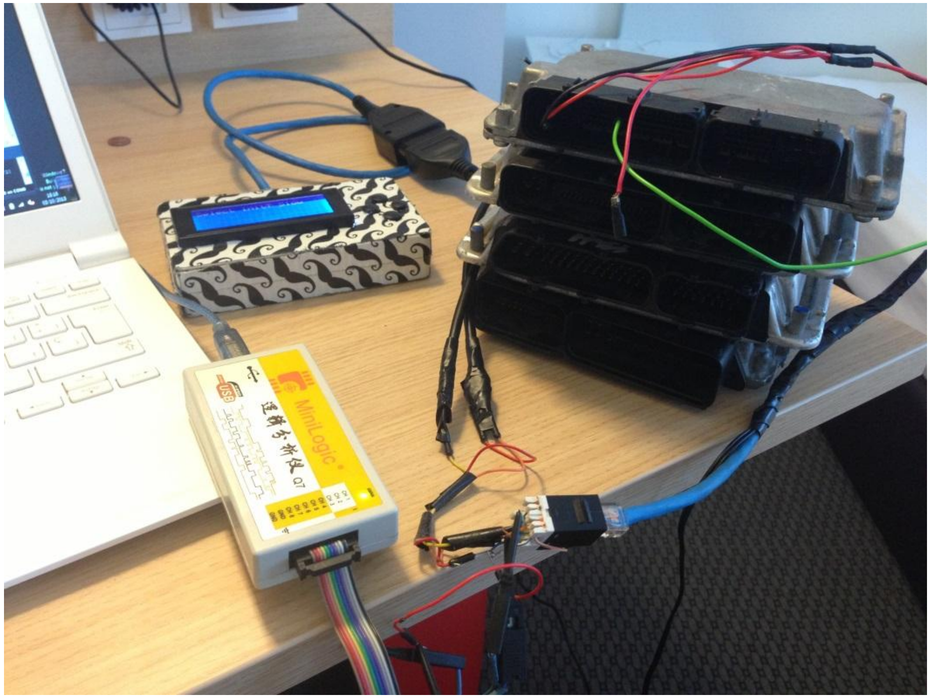


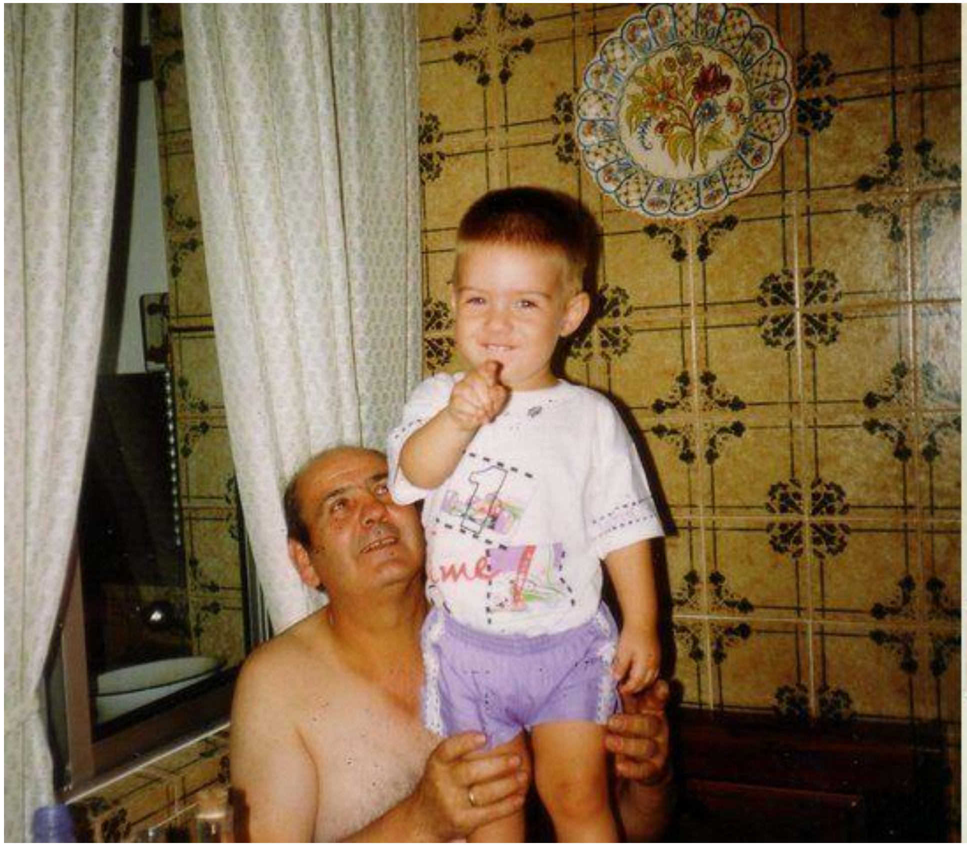
Alberto Garcia Illera (@algillera)
Javier Vazquez Vidal (@fjvva)

Javier Vázquez Vidal

- Hardware Security specialist
- Loves breaking “toys” security
- From Cádiz (Spain)



Alberto García Illera

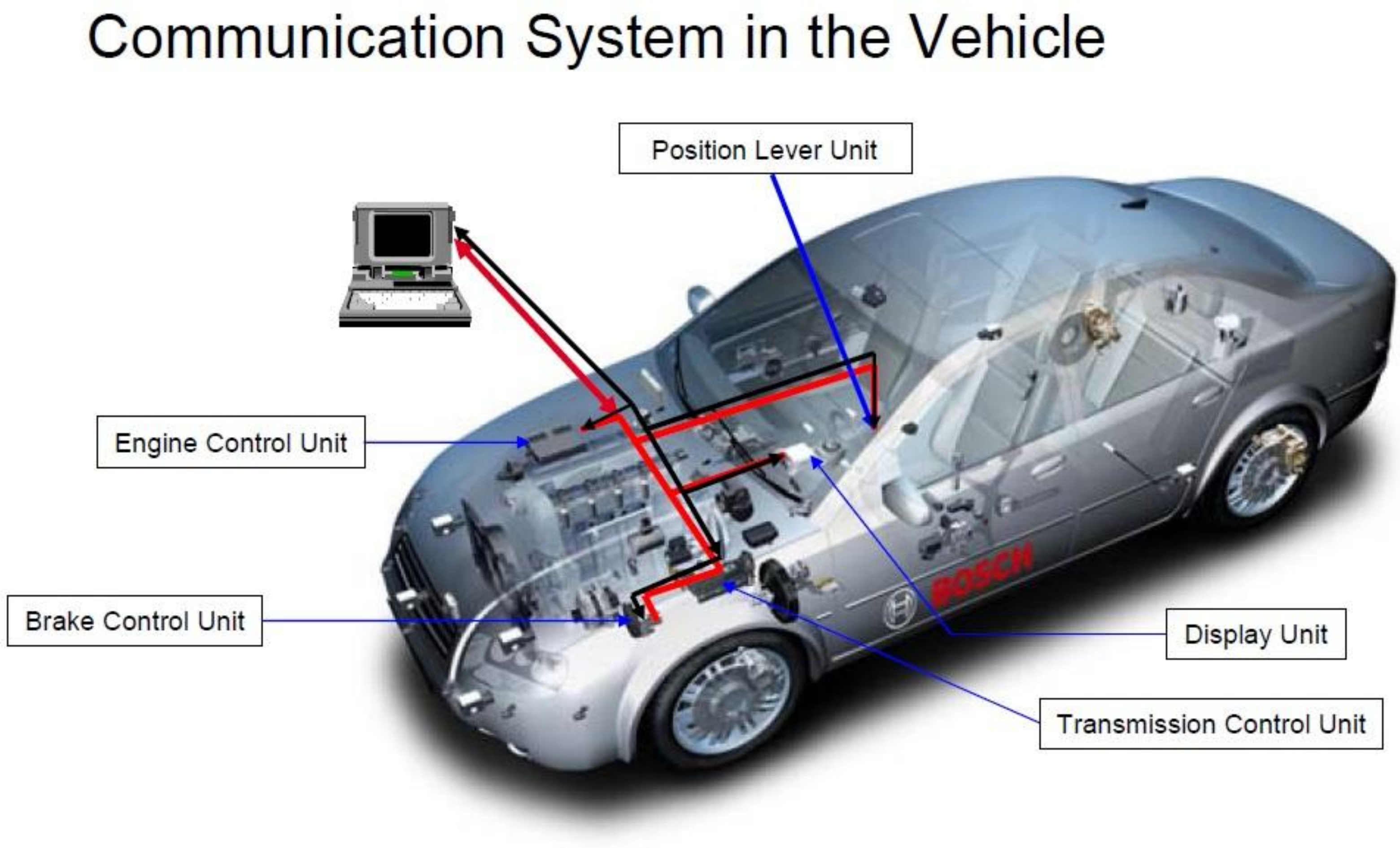


Where are we today?

Index

- Kline vs CANBus
- ELM327
- CHT Hardware
- Statistical Analysis
- CAN ID takeover attack
- Car forensics

Vehicle Electronic Control Units



Vehicle Electronic Control Units

- Each ECU uses and accepts a set of unique IDs (addresses) in the network.
- They have authentication/encryption protection against non authorised (dealer) access.
- Data is usually stored in them for diagnosis/forensincs purposes, aswell as for car behaviour (configuration).

Vehicle comunication protocols

CAN Bus (ISO 11898) features

- Mandatory for vehicle communications since 2008
- 1Mb/sec max speed (for CAN 2.0 network)
- Practically immune to noise
- Requires “expensive” hardware on physical layer

NO

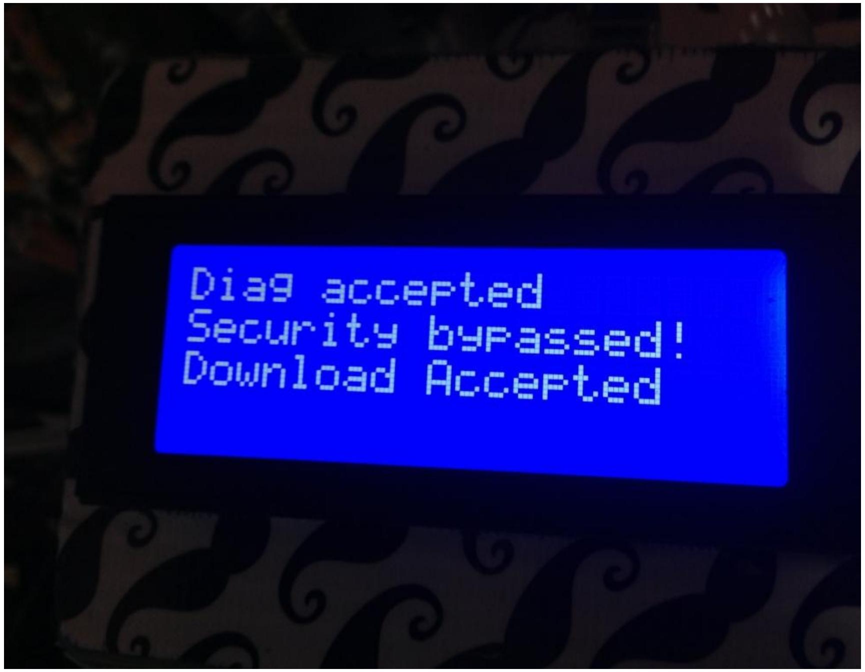
K-Line (ISO 9141-2) features

- Exists on vehicles dated from 1998 – 2010
- Works on 12VDC
- Max speed of 250Kb/sec (where available)
- Requires inexpensive hardware for physical layer

ECU Tool K-LINE library V2

It was released with the idea of allowing everyone to use the ECU tool for developing their own applications with the same hardware in a flexible way. Has the following features:

- Holds the most common commands, and provides an easy way to use them.
- Includes LVL1, LVL3 and LVL41 security access algos.
- It is used for development of the ECU tool main FW, so it will be updated frequently with new functions and bug fixes.



Some facts about CAN data

- In most cases, each different model, even from the same manufacturer, **uses different data** (commands/format) than the others, which makes it impossible to find an “**universal**” command to perform an action.

- It is usually **wrongly** stated that the CAN data is “plain”. It is formatted according to the manufacturer specifications, and most of the times cannot be parsed in the same way from two different vehicles.

- CAN is an **OSI level 1 and 2 protocol**, so data “security” cannot exist within it, and should be implemented by the user (manufacturer).

Some facts about CAN data

- There are three types of commands which can trigger an action: User, Diagnostics, and System commands.
    - **Diagnostics** commands cannot be captured by simply listening to the bus, as they are sent by diagnosis equipment (unless you capture them while the equipment is plugged and sending them). They usually allow triggering certain actions in order to “test” elements. They can be sent over the OBD2 port.
    - **System** commands are generated by events such as the ABS, and most of the times cannot be sent over OBD2 since they will get filtered by the gateway.
    - **User** commands are broadcasted when the user triggers an action, such as opening doors, turning lights on, opening windows, etc… Most of the times, they cannot be sent over OBD2, as they will be filtered by the gateway.

ELM327

The ELM327 is the most wide-spread interface for OBD2 Generic and Enhanced diagnostics. It is handled by AT commands, and works with several protocols:

- SAE-J1850 PWM (41.6 kbaud)
- SAE-J1850 VPW (10.4 kbaud)
- ISO 9141-2 (5 baud init, 10.4 kbaud)
- ISO 14230-4 KWP (5 baud init, 10.4 kbaud)
- ISO 14230-4 KWP (fast init, 10.4 kbaud)
- ISO 15765-4 CAN (11 bit ID, 500 kbaud)
- ISO 15765-4 CAN (29 bit ID, 500 kbaud)
- ISO 15765-4 CAN (11 bit ID, 250 kbaud)
- ISO 15765-4 CAN (29 bit ID, 250 kbaud)

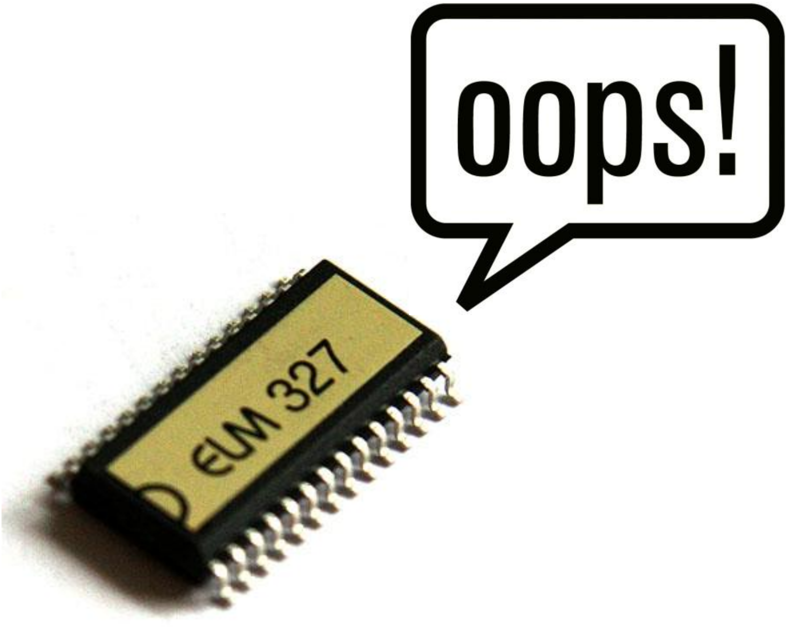
ELM327

Enhanced diagnostics allow the use of OEM specific commands (when known), and therefor, having some fun!

Since it works on AT commands, only a serial interface is required to handle it.

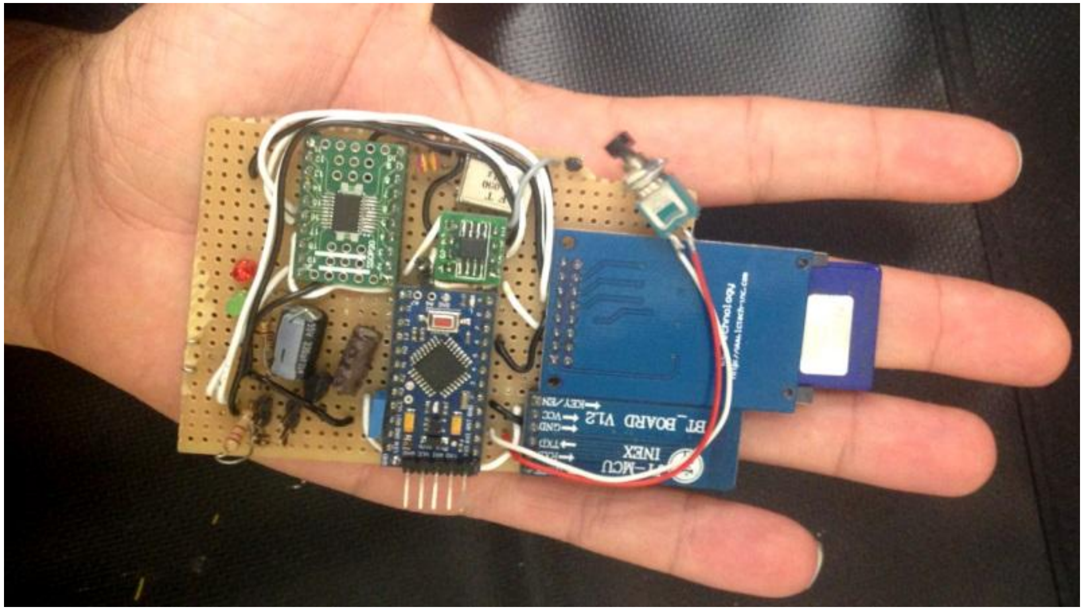
But, what happens when:

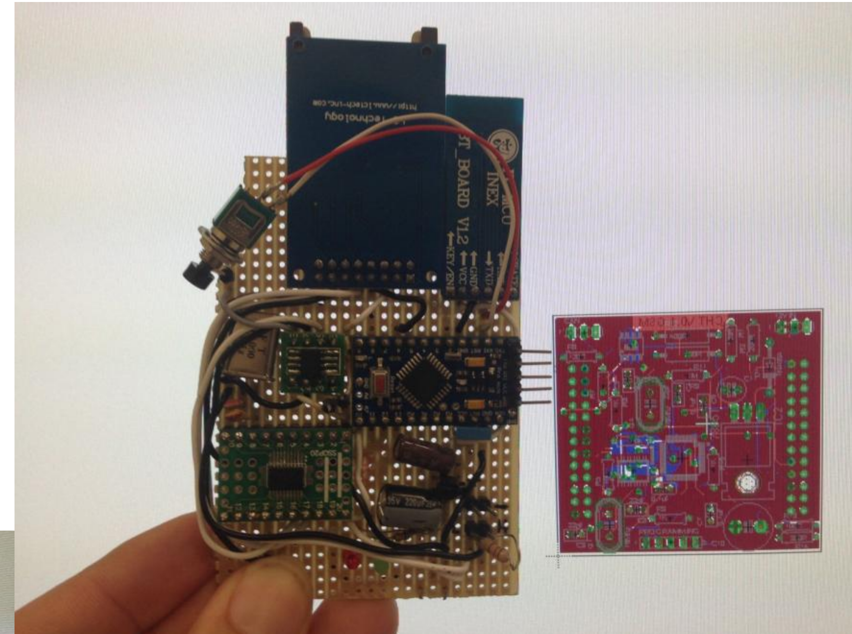
- The speed of the CAN bus you want to work with is 125kbaud or 800kbaud?
- We need to use “advanced” stuff?

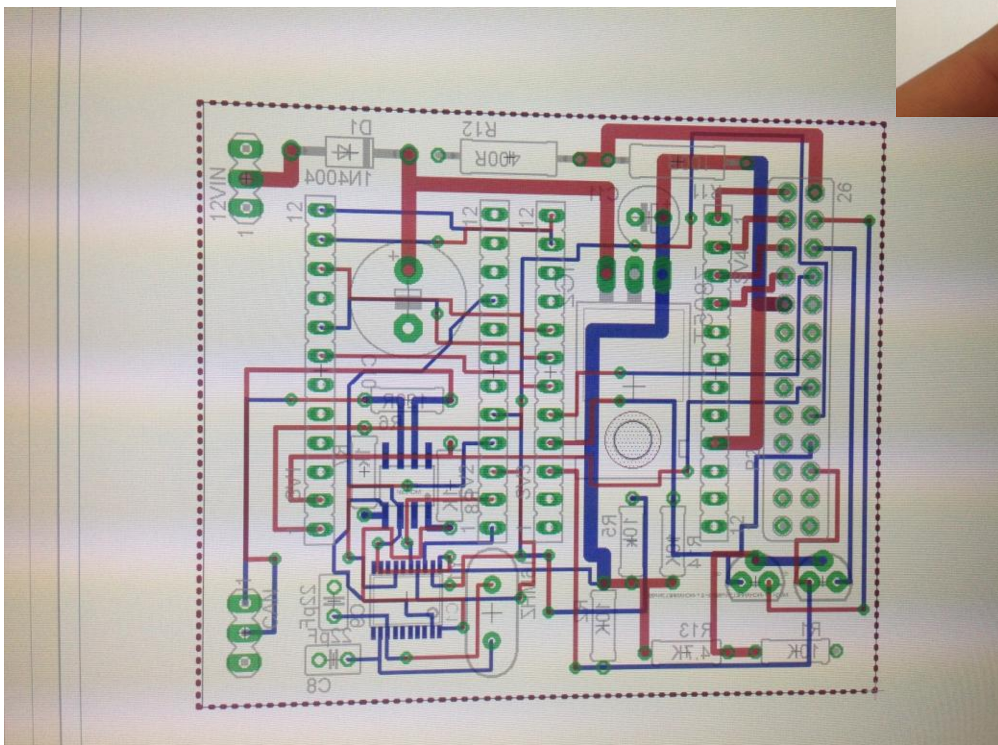


CHT, the Can Hack Tool

- What is it?
  - It is a piece of hardware based on the Arduino Mini Pro, that acts as an interface between the system (CAN bus target) and the user.
  - Its hardware is still under development, but an alpha POC version is already being used for firmware development.







CHT, the Can Hack Tool

- What does it do?
  - Fully field configurable via menus (filters, masks, speed…).
  - Has option to save configuration on internal EEPROM to load it at startup.
  - Capture and inject CAN data (no one would have guess that, right?). Capture has a success rate of 92% of the frames for a SD log file on a fully used bus (no interframe space) at 500kbps.
  - Inject errors and manipulate the frames on realtime.
  - Generate an active IDs report.
  - Perform basic analysis of the data being transmitted on the bus.
  - Store data logs for exhaustive analysis by sending them to a Back End.
  - Firmware can be updated in the field to add/change functions (requires a second Arduino Mini Pro).

CHT, the Can Hack Tool

- Hardware overview
  - 1x Arduino Mini Pro - $2,8USD
  - 1x Wireless module (Bluetooth, Wifi or GSM) - $6,5USD for BT
  - 1x MCP2515 DIP-18 CAN controller -$4,5USD
  - 1x MCP2551 CAN driver - $2,8USD
  - 1x SD Breakout board+SD card (class 4 or higher) - $5USD (2GB)
  - 1x 12VDC to 5VDC regulator(78L05)+components - $4USD

Total: **$25.6USD** (Sorry, things got a little bit expensive ☹)

But…

- If in each car the commands have different IDs and/or payload, how I am going to know what is the ID for opening the doors in my car?

CAN Statistical Analysis

- What is it about?
  - Checking for the type of data (commands, performance feedback, component status...)
  - Determining packet structure (counters, in-frame checksums, different status byte possibilities..)
  - Determining frame frequency (5ms, 10ms…)
  - Determining if it is a command (sent once or for a short period) or a broadcast (timed)

CAN Statistical Analysis

- This kind of analysis vastly improves the comprehension of the data being transmitted on the bus
- The data that is transmitted might be different on each vehicle, but the methodology is almost similar
- By doing an statistical analysis to a CAN capture, we can determine what ID corresponds with a specific command in any car with a high rate of success.

CAN Statistical Analysis

- Why does it work?
  - Users have common habits. Probably, one of the first frames sent will be the door unlock/alarm disable command, and it will only be sent once.
  - The “open doors” command will probably be sent once, while the “open window” will be repeated while the action is taking place, but only on a small period of time.
  - Some values will only differ from zero once the engine starts or the car starts moving, which allows identification of, at least, what kind of parameter it might be (engine, chassis…).
  - We can always wake up the bus and check for packets that exist without commands, which will ease the filtering when the bus is awaken by the user actions.

CHT Webservice

- Due to the limitations of the hardware used (8bit processor, 30kB Flash, 2kB RAM), we created a website for our tests, and it will be made public so anyone can upload their captures to perform an analysis.
- <u>The system is still under development, and will be improved over time.</u>

- The current analysis has four different sections:
  - List with one time IDs present in the capture (possible commands)
  - Periodicity of active IDs.
  - Packets sent over a timeline
  - Possible match with some commands (still under development)

DEMO

Attacks on the CAN bus

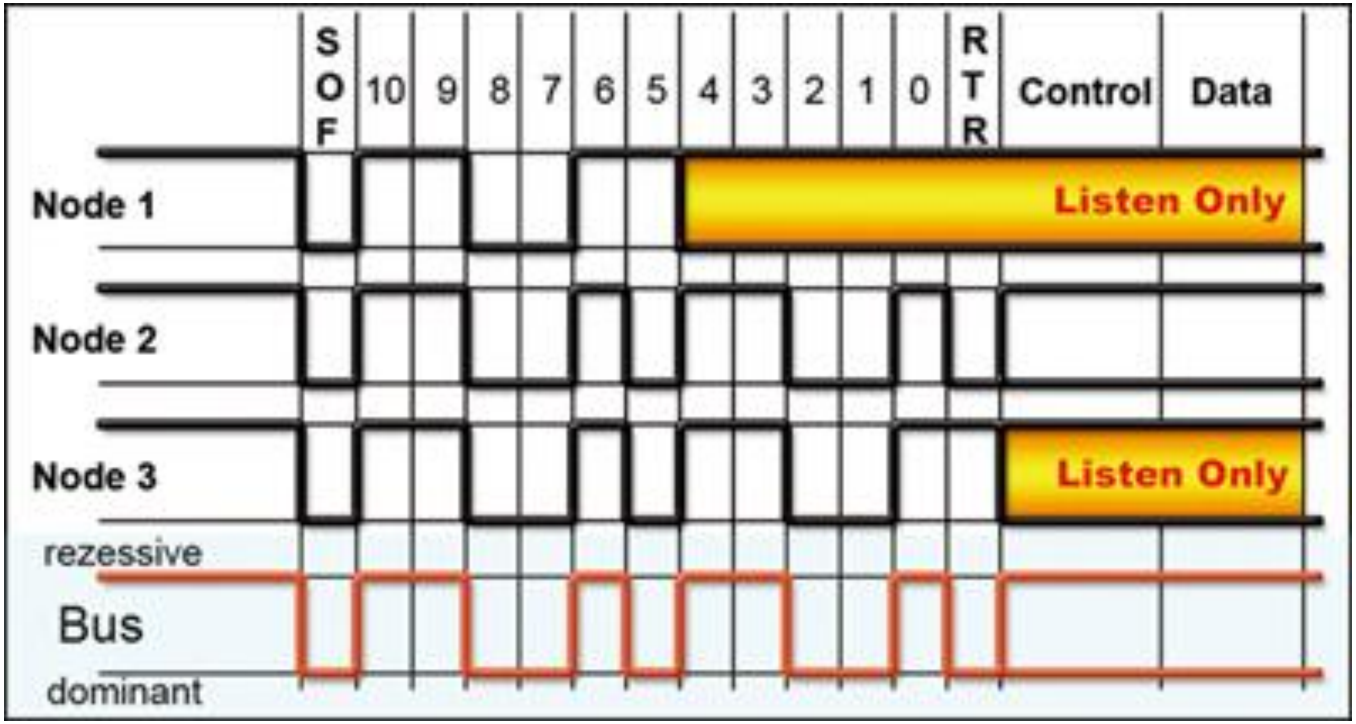
- When are they useful?
  - Some status messages are being broadcasted on a time schedule (every X milliseconds), so if you simply inject a different payload, it will be overridden by the original frame, making the injection useless.
  - Some devices require a condition (status) to be met in order to accept a commands, so forcing this status can allow to manipulate items under circumstances that normally you would not be able to.

Attacks: ID takeover

- How does it work?
  - We select a target ID we want to take over
  - We get attached to the CAN bus with a level shifter, and no protocol handler (just a mcp2551 f.ex), being able to directly read the bits that are transmitted real-time
  - Once we see that the target ID is transmitting, and we check the payload length, we can manipulate the frame to invalidate it in many different ways
  - After the original frame is invalidated, we inject our own frame using the same ID with a protocol handler (MCP2515)
  - We invalidate all future error and/or data frames from the target ID, and keep on injecting our own frames. If the attack is error generation oriented, the target module will give up transmitting after a while, which will allow to completely take over the ID without any effort.

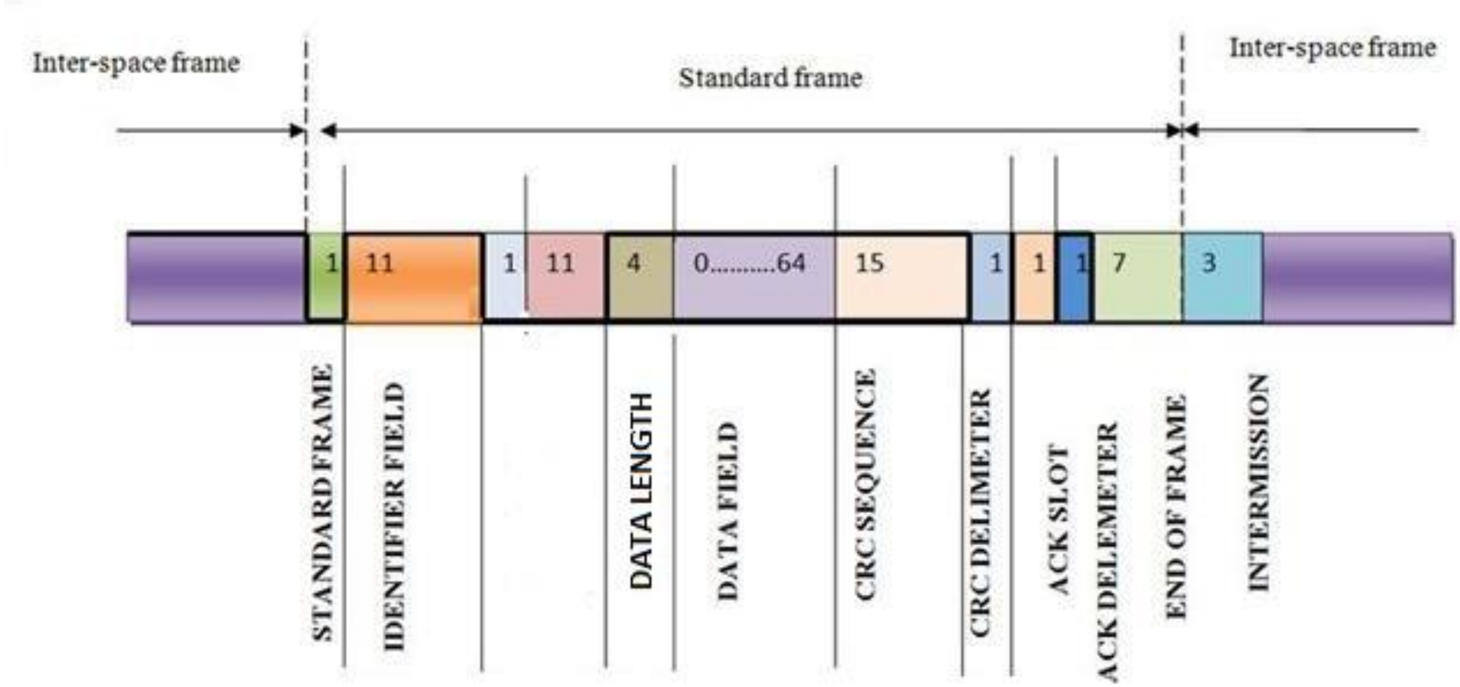
Attacks: ID takeover

- Why does it work?
  - Due to the arbitration system requirements, the bus can be driven low anytime (dominant), but not high (recessive).



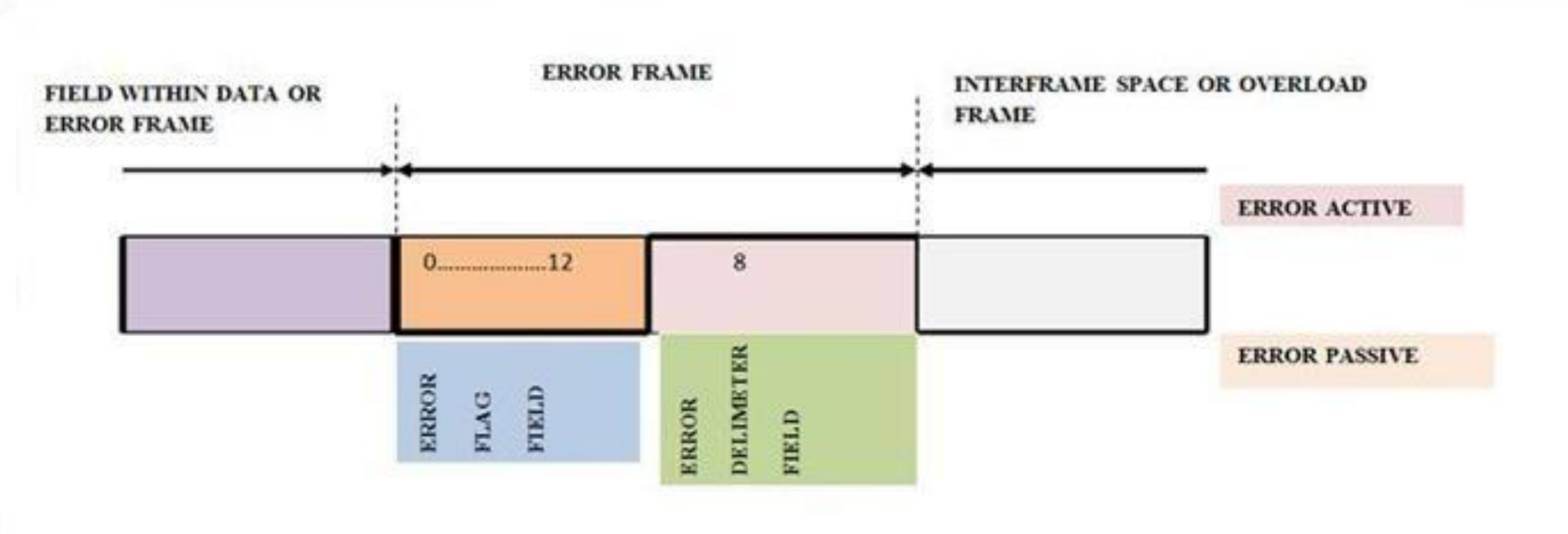
Attacks: ID takeover

- Why does it work?
  - The CAN frame structure allows to determine the ID (Identifier) and payload size (Data length) before the transmission is finished, so it gives you time to alter the data while it is still being transmitted



Attacks: ID takeover

- Why does it work?
  - The error detection system implemented in the CAN does not contemplate this kind of "errors" (non standarised), so with proper actions, none will be perceived by any module.



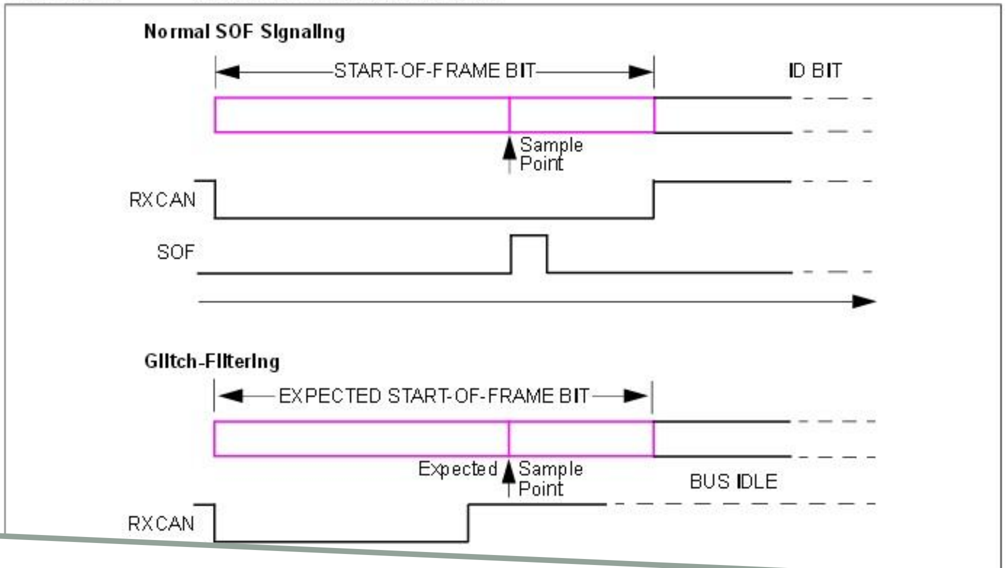
Attacks: ID takeover

- Why does it work?
  - The module sending the frame is responsible for error detection/correction, so if it does not make it (or the error frame is not successfully broadcasted), no errors are registered by any other module.

Attacks: ID takeover

- Why does it work?
  - The MCP2515 has the possibility to signal the detection of a SOF (Start of frame), making it easier for the MCU to perform the real-time analysis.

FIGURE 4-1: START-OF-FRAME SIGNALING

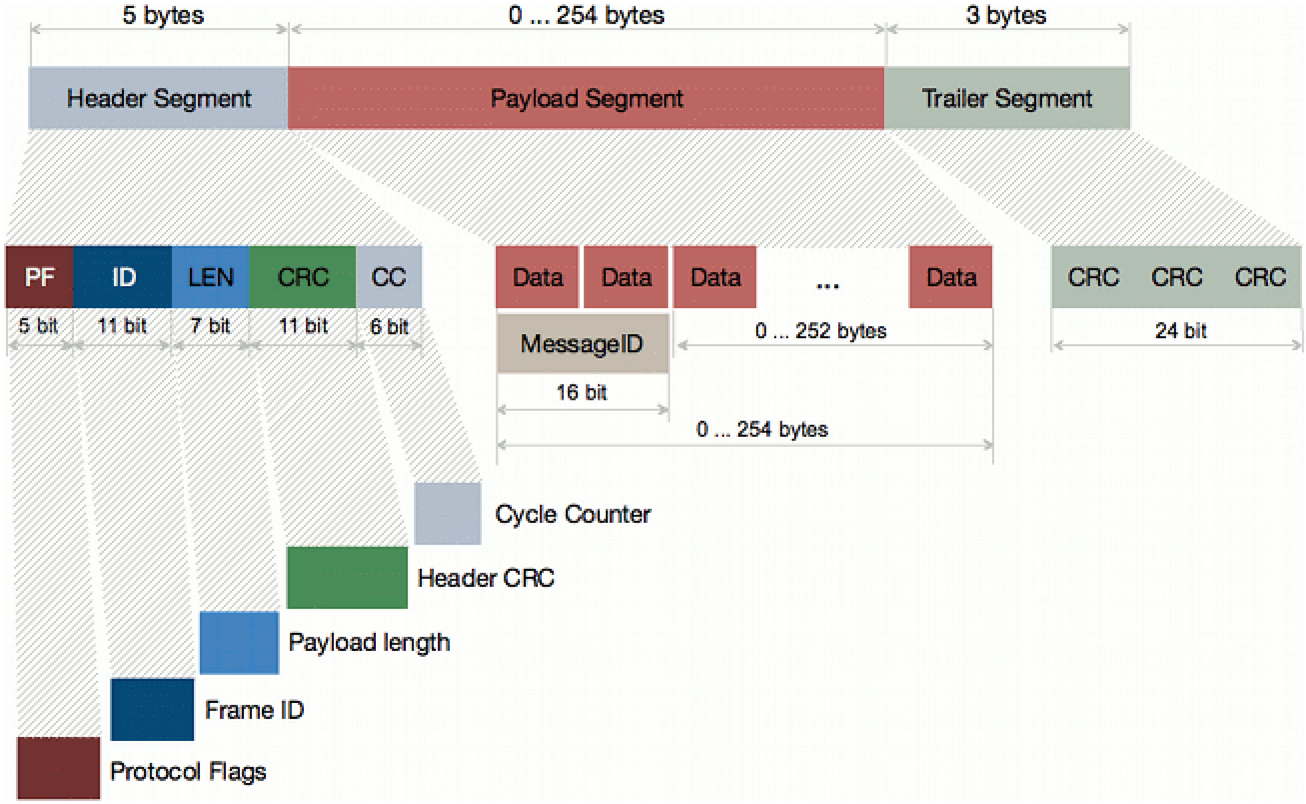


No longer required

DEMO

Did you know that…?

- Flexray:
  - Next gen protocol, up to 10Mbps.
  - Same working principle as CAN (Arbitration)



What was the reason? Forensics

Related information

- Most of the cars, since 1994, have a Crash Data Retrieval (CDR) function that stores several data related to the crash.

- It’s similar to the Black Box used in airplanes

- It stores information before and after the crash

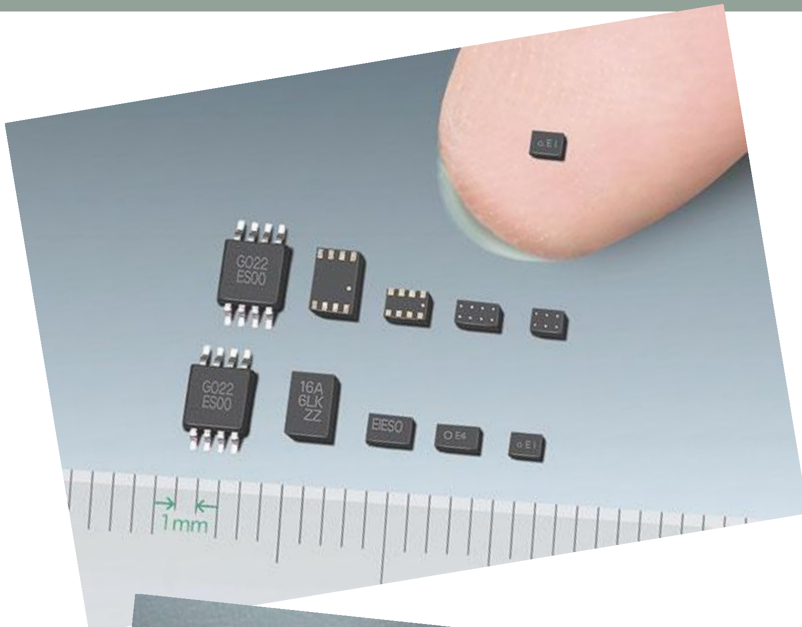
- This info is related with speed, RPM, brake use, ABS activity, accelerator pedal position (%)…

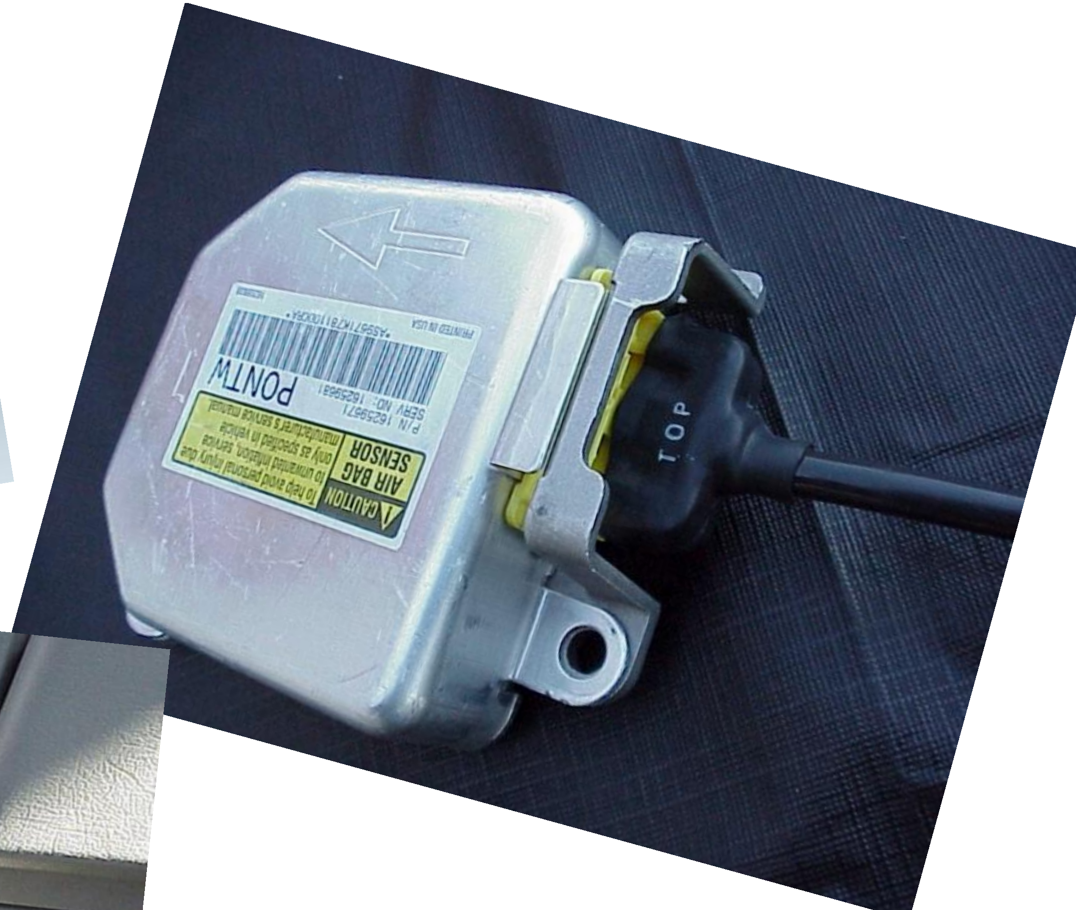
Where is the data?

- Almost all the cars store the crash data in the airbag ECU
- Usually this info is stored in a EEPROM (non volatile) memory
- There is costly hardware and software that must be used to retrieve and interpretate this information

The official hardware

- There is a official and expensive hardware/software from BOSCH to extract and parse the infomation
- There are three ways to connect the hardware with the Airbag ECU to retrieve the information:
  - Connecting to the ODB port (Authentication required)
  - Connecting with the airbag module (Authentication required)
  - Read directly the EEPROM memory (**No** authentication required)


G022
ES00
16A
6LK
ZZ
EIESO
1mm


CAUTION
AIR BAG
SENSOR
To help avoid personal injury due
to unwanted inflation, service
only as specified in vehicle
manufacturer's service manual
PONTW
TOP

How to extract the data of a ECU?

- The software/hardware to use in the BOSCH ECU is called CDR
- The “CDR Premium Tool Hardware Kit” costs **$8999**

What about poor guys?

- The software can be downloaded totally free
- So the code about parsing the data it's just in front of us…

CDR Buffer Overflow exploit

Supported Vehicles

Click on any of the supported vehicle links below to get started:

Acura	Hummer	RAM
Buick	Infiniti	Rolls-Royce
BMW	Isuzu	SAAB
Cadillac	Jeep	Saturn
Chevrolet	Lancia	Scion
Chrysler	Lexus	SRT
Dodge	Lincoln	Sterling
Fiat	Mazda	Suzuki
Ford	Mercury	Toyota
Geo	Mitsubishi	Volkswagen
GMC	Nissan	Volvo
Holden	Oldsmobile	
Honda	Pontiac	

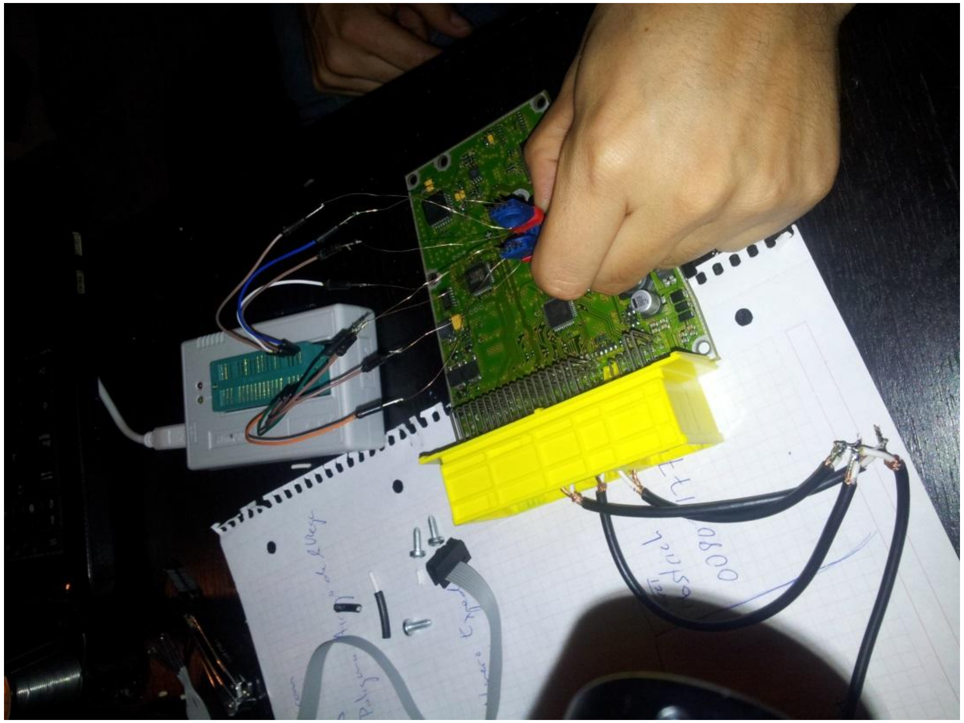
Mercedes??

Once upon a time

- A client contacted us to do a forensic job into a car that was not supported by the CDR tool (Mercedes)
- Our face was something like:

What we did

- We did it the straight way: reading directly from the EEPROM memory



What’s next?

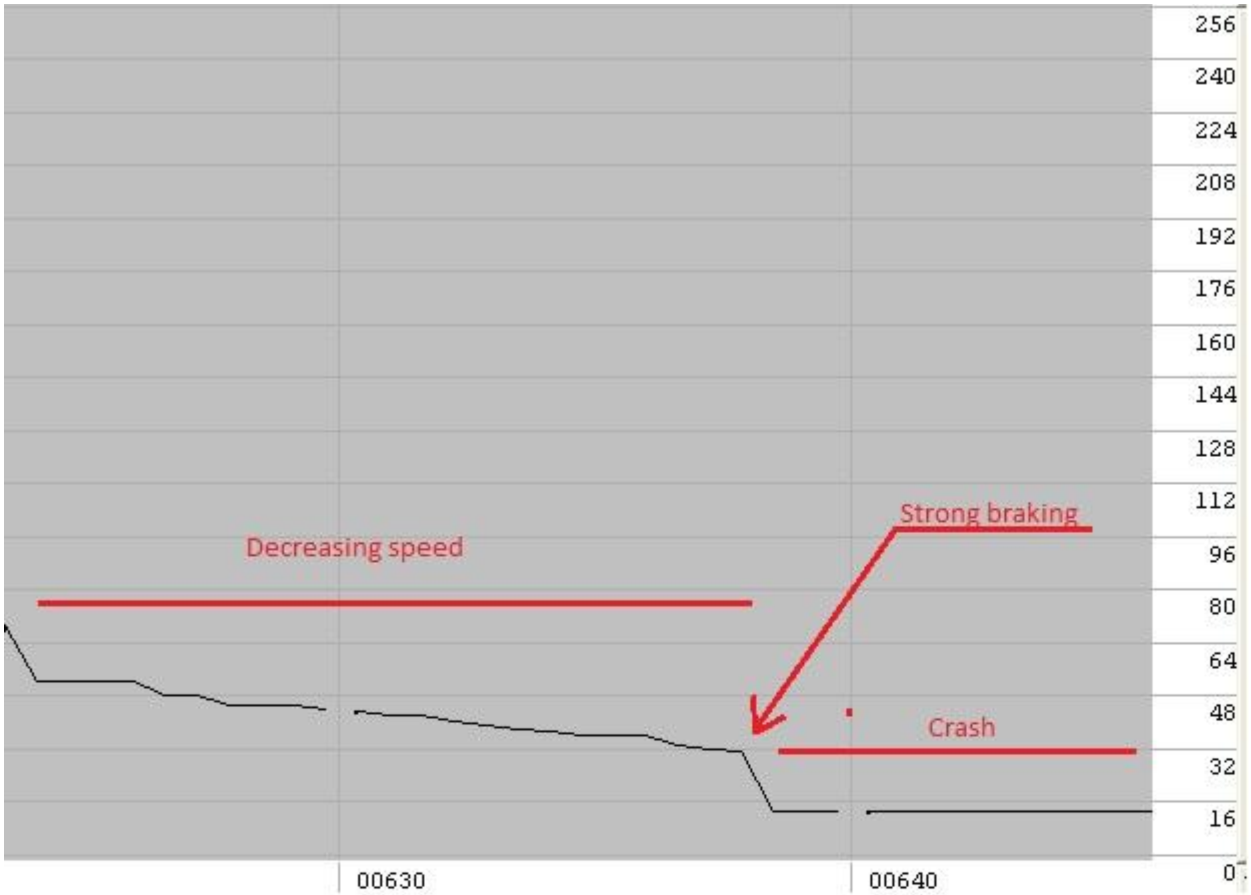
- We retreived all the information stored in the EEPROM memory but we didn’t have a way (with his wife) to parse it because the CDR program did not support Mercedes at that time
- So, we used a tool to reset the crash file data and doing after that a bindiff
- Doing this we knew what parts of the binary had changed and so we knew these parts of the full binary contained info about the crash

...

- The next step to do with the already filtered data was looking for the speed of the car at the moment of the crash
- To do this we used WinOLS to view the graphs and be able to distinguish between the crescent and descrescent graphs
- The sorting was made because the speed in a car crash is always descrecent (Captain Obvious strikes again!)

We had a match!!

- After doing this we found a interval with values that could match with the speed in a car crash



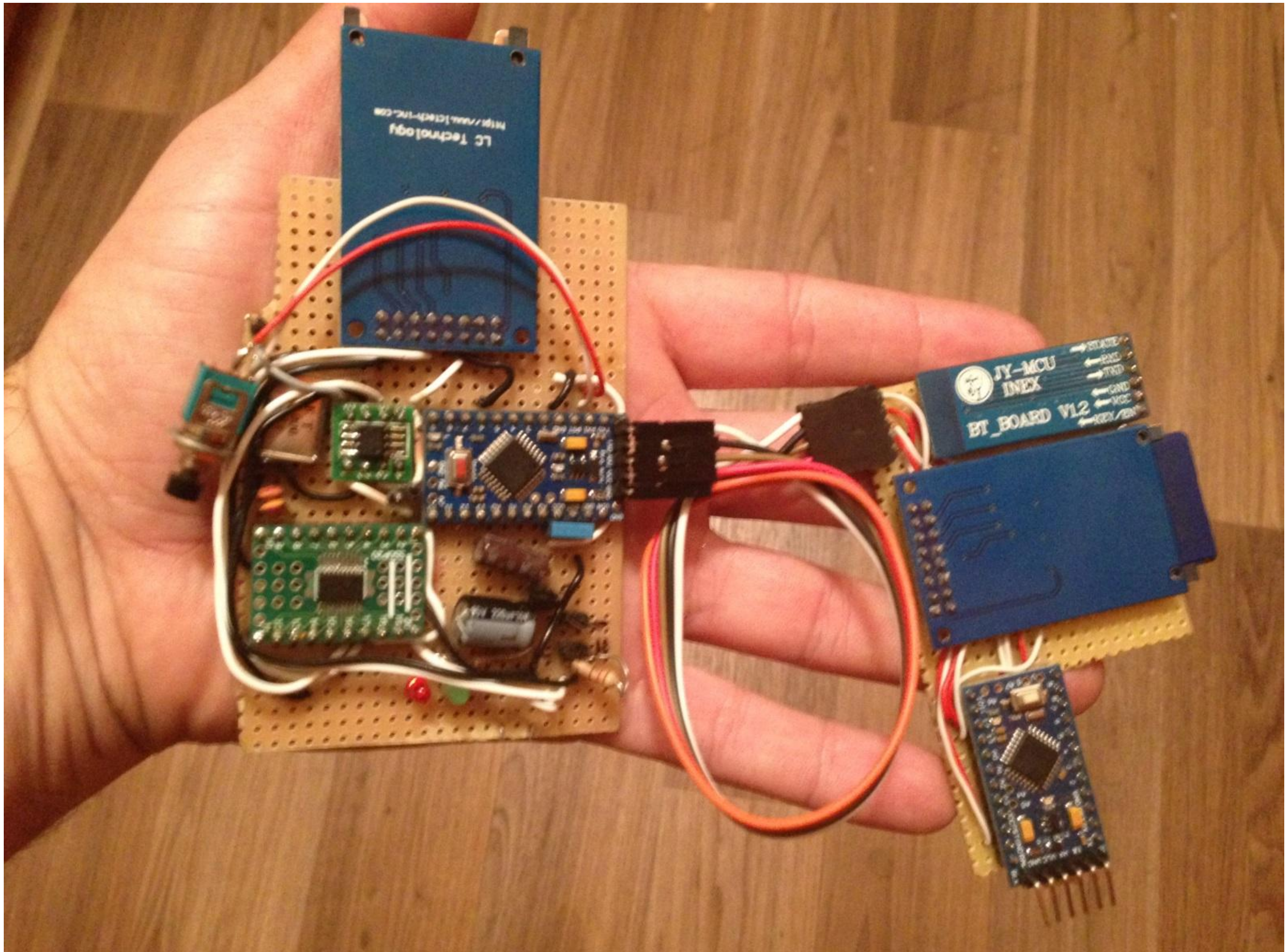
But that’s not all!

You should probably look forward to the release of the ECU tool V3 HW/FW, as the following features will be added:

- CAN interface
- EDC16 support via CAN
- Crash data retrieval (Still on the works, but getting closer!)
- CHT emulation (with basic functionality)
- Optional OTA FW update via BT.

Cost of the upgrade, including OTA, will be around $10.

Keep a close look at our GitHub repo: github.com/fjvva


LC Technology
JY-MCU
INEX
BT_BOARD V1.2

Thank you

- All of you for being here today
- To our family and friends. They are always there when we need them
- All those who want to understand how and why things work

- **Alberto Garcia Illera (@algillera) agarciaillera@gmail.com**
- **Javier Vazquez Vidal *(@fjvva) javier.vazquez.vidal@gmail.com***